

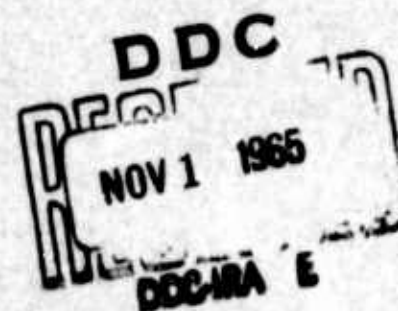
AD622767

COMPUTATIONAL EXPERIENCE WITH THE
BALAS INTEGER PROGRAMMING ALGORITHM

Raoul J. Freeman

October 1965

CLEARINGHOUSE FOR FEDERAL SCIENTIFIC AND TECHNICAL INFORMATION			
Hardcopy	Microfiche		
\$1.00	\$0.50	13 pp	40
ARCHIVE COPY			



P-3241

COMPUTATIONAL EXPERIENCE WITH THE BALAS INTEGER PROGRAMMING ALGORITHM

Raoul J. Freeman^{*}

The RAND Corporation, Santa Monica, California

I. INTRODUCTION

A recent and significant addition to the literature of partial enumeration methods of solving integer linear programming problems has been the algorithm of Balas⁽¹⁾. Building upon the foundation laid by (1) and (2), a version of this algorithm was outlined in (3) that especially lends itself to computer implementation. This has since been programmed, and the purpose of this report is to describe some computational experience with the algorithm and to discuss future avenues of research that may be undertaken.

*Any views expressed in this paper are those of the author. They should not be interpreted as reflecting the views of The RAND Corporation or the official opinion or policy of any of its governmental or private research sponsors. Papers are reproduced by The RAND Corporation as a courtesy to members of its staff.

II. SOME ASPECTS OF THE ALGORITHM

We shall first review certain features of the algorithm. For a detailed exposition see (3). Our problem is to

$$\text{Minimize} \quad \sum_i c_i x_i$$

$$\text{Subject to} \quad b_j + \sum_i a_{ij} x_i \geq 0 \quad j = 1, \dots, m$$

$$x_i = 0 \text{ or } 1 \quad i = 1, \dots, n$$

$$c_i, b_j \text{ and } a_{ij} \text{ are constants } (c_i \geq 0)^*.$$

Given that there are upper bounds on the integer variables in question, the above representation is sufficiently general to describe any integer programming problem. A solution is an n vector of 0's and 1's. There are obviously 2^n different possible solutions to the above problem. A feasible solution is a solution which satisfies $b_j + \sum_i a_{ij} x_i \geq 0$, $j = 1, \dots, m$, and an optimal feasible solution is a feasible solution which yields the lowest value for $\sum_i c_i x_i$.

The basic idea of the Balas algorithm is to obtain an optimal feasible solution (or knowledge that none exists) without having to evaluate each of the 2^n possible solutions. However, the procedure does enumerate a subset of the 2^n solutions, and there is no guarantee that the size of this subset will not approach 2^n for any given problem. Thus, as with any enumerative procedure, computational experience is the best indicator of its worthiness for various types of problems.

*Any nonbinary integer variable x with upper bound v has the binary representation $x = \sum_{i=0}^k 2^i y_i$, where k is the smallest integer such that $v \leq 2^{k+1} - 1$ and the y_i are binary variables. The transformation $x = 1 - y$ will change any $c < 0$ to one that is > 0 .

A partial solution of size k is a specification of values (0 or 1) for k of the n variables. A completion of a partial solution is a specification of the other $n-k$ variables not included in the partial solution. Thus, if it can be shown that a certain partial solution has no feasible completion, or if a completion which gives the least value to $\sum_i c_i x_i$ can readily be found, then one may eliminate 2^{n-k} possible solutions from consideration in searching for an optimal feasible solution to the original problem. The Balas procedure consists of considering a sequence of partial solutions, and of trying to eliminate groups of completions by showing that none are feasible or that an objective function value as good as any contained in that group of completions has already been attained. The algorithm proceeds in an exhaustive and nonrepetitive manner which leads to the implicit enumeration of all possible solutions.

Fathoming a partial solution is defined as: (a) finding a best feasible completion which yields a lower value for the objective function than the best feasible solution known to date; or (b) showing that no feasible completions exist which yield a lower value to the objective than the best feasible solution known to date. The procedure begins with a partial solution where $k = 0$. A best completion (i.e., that completion yielding the smallest objective function value) is the one with all $n-k$ variables being set to zero (remember all $c_i \geq 0$). If this best completion is feasible, then it is not necessary to consider other possible completions of this same partial solution. If it is not feasible and it also cannot be shown that there are no other feasible solutions which yield lower objective function values than the best feasible solution known to date, then a new partial solution is contemplated. The new partial solution (k raised to $k + 1$) is obtained via an augmentation procedure. Obviously, if fathoming occurs at $k = 0$, then all 2^{n-0} possible solutions have been implicitly enumerated. If fathoming occurs at $k > 0$, then a smaller number of solutions have been implicitly enumerated, and the rest must still be examined.

The augmentation procedure that is used in the present version of the algorithm is to add that variable which will decrease the infeas-

ibility of the present solution the most. This tends to motivate feasible solutions to appear early. Once a feasible solution is obtained, it is kept in storage and is replaced only by another feasible solution which yields a lower objective function value. Thus, the enumeration is "primal feasible" and termination prior to attaining optimality still may lead to a fairly good solution.

All partial solutions must either be fathomed or augmented. From the point of view of minimizing the total amount of computation, it behooves us to try to get the fathoming done with as little augmentation as possible.

Any partial solution (except where $k = 0$) that is being fathomed can itself be viewed as an augmentation of other partial solutions. Consider a partial solution of size k , and assume that an augmented partial solution of size $k + 1$ can be fathomed with the value of the additional variable $= 0$ and then can also be fathomed with the value of the additional variable $= 1$; then we have also fathomed the partial solution of size k . This can readily be seen because the fathoming of the augmented partial solution produces 2^{n-k-1} implicitly enumerated possible solutions for each of the two possible values of the additional variable. Thus, we have a total of $2(2^{n-k-1}) = 2^{n-k}$ implicitly enumerated solutions which is the required number for fathoming of the partial solution of length k .

III. COMPUTATIONAL EXPERIENCE

Unfortunately, at present, there is no standard group of test problems against which integer programming algorithms can be tested. However, a group of test problems have been gathered in (4) with which three "cutting plane" codes are compared. The results are tabulated in terms of number of iterations required by each. Running times are not given, although this would provide the most convenient way to compare the reformulated Balas algorithm with the cutting plane methods. Thus, we shall report our running times on certain of the problems of (4) without attempting code comparisons.

There are ten "fixed charge" problems in (4). These problems feature special constraints which force certain variables to assume nonzero value if some of the other variables take on nonzero values. Our experience with the algorithm applied to some of these problems is summarized below.

Haldi Problem No.	No. of Binary Variables	No. of Inequalities	Running Time To Termination (seconds)	Running Time To Optimal Value of Obj. Function (using upper bound on obj. func.)
1	14	4	3	3
2	14	4	3	3
3	14	4	4	4
4	14	4	2	2
5	20	6	420	242
6	20	6	120	68
7	20	4	360	235
8	20	4	60	1+

The last nine problems in (4) were obtained from IBM and are a potpourri of integer problems which feature matrices of 0's and 1's. Listed below is our experience with some of these problems.

IBM Problem No.	No. of Binary Variables	No. of Inequalities	Running Time To Termination (seconds)	Running Time To Optimal Value of Obj. Function (using upper bound on obj. func.)
1	21	7	34	4
2	21	7	20	2
3	20	3	2	2
4	30	15	>600	60*
6	31	31	>600	2**
9	15	50	144	1+

In another problem of the "fixed charge - knapsack" variety with 48 constraints and 26 variables, the Balas algorithm did not find an optimal solution in 10 minutes. However, although not terminating it did reach the optimal value of the objective function in 30 seconds when supplied with an additional constraint forcing the objective below a certain initial value.

On a problem from (5) where the PKIP91 code did not obtain an optimal solution in 20 minutes of 7090 time, we did find an optimal solution in 8 minutes. However, we did not obtain termination. In other words, the algorithm did not know it had found the optimal solution and had to continue its search. The problem was 31 x 27. Introduction of an initial upper bound on the objective function did cut the time to reach an optimal solution to a few seconds. If a good estimate of the value of the optimal solution is available a priori then the computation could be terminated as soon as it achieves that value.

Tentative conclusions that can be drawn from the above experience are as follows:

1. The algorithm seems fairly successful in its present form for problems of less than 30 variables, but may not be able to efficiently handle problems containing a greater number of variables.

2. The algorithm seems capable of handling large numbers of constraints given that the number of variables is kept limited.

* Optimal value of objective function = 10; in time recorded Balas achieved 12.

** Optimal value of objective function = 18; in time recorded Balas achieved 19.

3. The algorithm tends to produce feasible solutions in short order, and thus seems appropriate for large problems where only feasible, and not optimal, solutions are sought.

4. Prior knowledge about likely values for variables and bounds on the objective function can be extremely useful. Our results so far suggest that ad hoc guessing procedures about such values combined with this algorithm may be of great interest.

IV. FURTHER DEVELOPMENTS

The algorithm was originally programmed in essence following the outline of reference (3), and most of the computational experience reported herein is based on that version. Certain modifications, described below, suggested themselves and some of these have been incorporated in a second version of the algorithm that has now been programmed.

1. Some computation time can possibly be saved by setting a number for the first "acceptable" value of the objective function. In other words, we are indicating an upper bound beyond which a feasible solution is not of interest. This, in essence, imposes a constraint that the objective function shall be less than a certain value. This eliminates explicit consideration of relatively unfavorable feasible solutions. The upper bound will be an input parameter.

2. On some problems it was found that the algorithm produced an optimal solution quickly, but took overlong to terminate. To hasten termination, an aspiration level of the objective function can be explicitly introduced. Once this aspiration level is attained, computation is terminated. The aspiration level will be an input parameter.

3. In moving from one solution to the next, the only criteria was that there be an improvement in the objective function. An input parameter will be introduced to allow a specification of the minimum size of improvement that is acceptable in moving from one feasible solution to the next. This is intended to speed progress toward an optimal solution.

4. In picking the variables to augment a partial solution, primary attention is given to increasing feasibility. Another computational parameter has now been introduced which allows some consideration of enhancement of the objective function in picking augmentation variables.

5. Provision has now been made so that any initial solution can be used as a starting point. Thus, a priori information can be utilized (see reference 3, Sec. V, for fuller discussion).

Preliminary evidence indicates the upper bound and the aspiration level are having the most pronounced effects.

Among the avenues of new research which remain open are the following:

(a) At present, the algorithm tries only to find the best completion of each partial solution, and then tests for its feasibility. Augmentation takes place if it cannot be shown that there does not exist a feasible solution yielding a lower objective function value than the incumbent feasible solution. Instead of finding merely the best completion, the algorithm could be adjusted to scan a few completions (the best, next best, etc.) in order to try to obtain the feasible completion that is best. This would consist of making an array of say the ten "nearest" best (in terms of objective function value) solutions and scanning these at every iteration.

(b) Feasibility considerations are determined at present by considering every constraint separately to determine if any of the variables in the T^S set can decrease the total infeasibility of the present intermediate solution to the problem. Use of other constraints, which must also be satisfied (composed of combinations of the original constraints), could lead to quicker determination of "no possible feasibility." This notion is analogous to the surrogate constraint concept put forth by Glover (2). This same notion could help in limiting the T^S set to more ultimately desirable augmentation variable candidates in terms of their effects upon groups of constraints. The idea of the surrogate constraint is to find something which "cuts" more strongly than one constraint at a time. We can also consider taking a partial solution as given and solving the remaining problems in "continuous" fashion. The values of the fixed variables would be substituted into the problem, and the problem would be solved as a regular (noninteger) linear programming problem with the free variables restricted to be between 0 and 1. Infeasibility of this problem indicates infeasibility of the integer problem with the given partial solution. This is a more powerful feasibility test than the one in the present algorithm.

(c) A series of binary variables are utilized to represent integer variables with upper bounds greater than 1. The columns of these binary variables are 2^i multiples of each other. This fact could be used to generate such columns whenever they are needed during the

course of the computation. Thus, only one column needs to be explicitly carried.* As the algorithm is programmed at present, no advantage is taken of the special structure represented by the binary transformations.

(d) Given that the algorithm seems to function well in the early stages of computation (i.e., finding an initial feasible solution and improving upon it), it might be well to start thinking about hybrid schemes, which would use this algorithm as a first stage and then switch to another technique to finish the computation.

*Originally suggested by G. W. Graves.

V. CONCLUSION

It is recommended that research on this algorithm be continued in order to establish its value of lack thereof in dealing with various types of integer programming problems. Also, it would be of value to run various codes upon the same set of problems in order to establish definite computational guidelines for practitioners in the field.

REFERENCES

1. Balas, E., "An Additive Algorithm for Solving Linear Programs with Zero-One Variables," Operations Research, Vol. 13, No. 4 (July-August 1965), pp. 517-546.
2. Glover, F., "A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem," Management Sciences Research Report No. 25, December 1964, Graduate School of Industrial Administration, Carnegie Institute of Technology. Revised for release February 1965.
3. Geoffrion, A., A Reformulation of Balas' Algorithm For Integer Linear Programming, The RAND Corporation, RM-4783-PR, September 1965.
4. Haldi, J., "Twenty-Five Integer Programming Test Problems," Working Paper No. 43, Graduate School of Business, Stanford University, December 1964.
5. Wagner, H. M., R. J. Giglio, and R. G. Glaser, "Preventive Maintenance Scheduling by Mathematical Programming," Management Science, Vol. 10, No. 2, January 1964, pp. 316-334.